A Parallel CAVLC Design For 4096x2160p Encoder

Huibo Zhong[1], Yibo FAN[1a)], Xiaoyang ZENG[1]

[1]State Key Lab of ASIC & System, Fudan University, Shanghai, China

[a)] fanyibo@fudan.edu.cn

Abstract — This paper presents a high performance VLSI design of Context-Based Adaptive Variable Length-Coding (CAVLC) for 4096x2160p@60fps H.264/AVC encoder. A parallel architecture is proposed to make the scan and encode stage work simultaneously. Four coefficients are scanned in parallel, and four *Levels* and *Run_before* are coded in parallel. From experimental results, only 120 cycles at most are needed to process one macroblock (MB), which reduce more than 50% cycles compare to state-of-the-art designs. The hardware implementation results show that the proposed design achieves real-time encoding at 250 MHz and the hardware cost is about 32k gates.

Keywords: *Entropy coding, CAVLC, H.264/AVC, Level coding*

I. INTRODUCTION

H.264/AVC [1] [2] is the latest international video coding standard. It significantly improves compression performance compared with previous standards since a lot of new technologies are used. H.264/AVC specifies two kinds of entropy coding methods that are Context-Based Adaptive Variable Length Coding (CAVLC) and Context-based Binary Arithmetic Coding (CABAC). CAVLC yields a higher coding efficiency than conventional VLC coding due to the context adaptive feature. On the other hand, because of the data dependency in CAVLC, it results in a complex encoding flow in hardware implementation.

As video resolution becomes larger, encoder need to support higher bitrate video stream. However, the entropy coding becomes the bottleneck of the whole encoder since its bit-serial processing is hard to speed up by increasing parallelism or pipelining. Many VLSI architectures for CAVLC were proposed [3-5]. The design in [3] is capable of processing 1080@30fps video in real time, it requires 500 cycles for high-quality applications and about 200 cycles for low bitrate applications to process one MB. Yi et al. [4] proposed a parallel structure and their work can process 1080@60fps video in real time, it requires about 323 cycles for some sequences for each MB. Recently, Literature [5] adopted a forward order for computing the CAVLC parameters and it takes 256 cycles to process 1080@60fps video. However, due to the complexities and data dependencies of CAVLC, it is hard to design an encoder to support high resolution applications such as 4096x2160p video or higher.

In this paper, we focus on increasing the hardware parallelism for CAVLC. A two-stage parallel CAVLC encoder is proposed to make the scan and encode stage work simultaneously. A paralleled scan engine which can scan four coefficients is proposed. In the same way, a paralleled *Levels* encoder is proposed to accelerate the speed of encoder engine. Four *Run_before* symbols are encoded simultaneously with *Levels*. As a result, only 4 cycles are needed by each stage of the design. Totally only 120 cycles at most are needed to process one macroblock (MB).

The rest of this paper is organized as follows. Section II briefly introduces the background of CAVLC. The proposed design is presented in section III. Section IV shows the performance analysis and comparison with previous works. Finally, conclusion is presented in section V.

II. BACKGROUND

For CAVLC coding, only the quantized transform coefficients of the residual images are coded. Other information, i.e., MB header, is coded using Exp-Golomb codes. The details of CAVLC can be referred from [6]. The coefficients need to be coded by CAVLC are listed as follows:

1) *Coeff_token*: The first VLC symbol in the CAVLC stream. It encodes both of the total number of nonzero coefficients (*Total_Coeff*) and the number of trailing ones (*TrailingOne*). *Total_Coeff* ranged from 0 to 16, and the range of *TrailingOne* is from 0 to 3. If the block has more than 3 trailing ones the rest are coded as normal coefficient. There are five tables used for *Coeff_token* encoding. The selection of the tables is according to the value of nC which is calculated according to the upper and left coded 4x4 block.

2) *Sign_trail*: The signs of the trailing ones. Each *Sign_trail* is coded into one single bit. 0 is assigned for positive ones and 1 is assigned for negative ones. The *Sign_trail* is coded in reverse zigzag scan order.

3) *Levels*: The nonzero coefficients excluding the trailing ones in reverse zigzag scan order are encoded as *Levels*. Each *Level* is encoded by one of the seven VLC tables depends on the magnitude of each successive encoded level.

4) *TotalZero*: *TotalZero* encodes the total number of zero coefficients proceeding to the first nonzero coefficient in reverse zigzag scan order for each 4x4/2x2 block. There are two separate VLC tables for the luma and chroma residual blocks.

5) *Run_before*: *Run_before* encodes the number of zeros preceding each nonzero coefficient in reverse zigzag order by a VLC table. There are two cases where the *Run_before* of a coefficient does not to be encoded: either there are no more zeros left or it is the last coefficient of which *Run_before* is left to encode.

978-1-4673-0219-7/12/$31.00 ©2012 IEEE

Compared with conventional VLC coding, CAVLC has higher coding efficiency due to the context-adaptive feature. However, this context-adaptive feature introduces many data dependencies, such as: 1) Many symbols (*Total_Coeff*, *Sign_trail* and so on) can only be obtained after the statistical process of each 4x4 block, which makes the scan and encode hard to be parallel. 2) For *Total_Coeff* coding, nC should be pre-determined, however, the value of nC depends on the upper and left coded 4x4 block. 3) For *Levels* coding, there are seven VLC tables, table selection depends on previous table number and the magnitude of the successive encoded *Level*. Due to the data dependencies in CAVLC, it is hard to design an encoder for high resolution, such as 4Kx2K or larger.

III. The Proposed Parallel Architecutre

3.1 2-stage pipeline, 4-coefficient parallel, cycle-balanced CAVLC encoder hardware architecture

Fig.1 shows the proposed two-stage CAVLC encoder. It includes two engines: scan engine and encode engine. For scan engine, the residual coefficients are fetched from the residual buffer in the backward order. And then the scan engine counts the coefficients such as *TrailingOne*, *Total_Coeff*, *TotalZero*, and so on. The *Run_before-level* symbols are extracted by the *Run_before -level* detector and stored in the Statistics Buffer. For encode engine, it encodes the coefficients generated by scan engine. All the code words produced by encode engine are packed into bitstream by Bitstream Packer.

From Fig. 1, both of the scan engine and encode stage may become the bottleneck of the encoder. Due to the serial processing of the symbol encoding in CAVLC, the cycles used for encode engine changes very much according to different bit rates.

Many previous works have adopted two-stage structure. Chen et al. [3] adopted a two stage structure, but their work did not implement coefficient-level parallelism, which makes it require at most 500 cycles to process one MB and greatly restrict the throughput. Yi et al. [4] also employed the two-way structure which process two coefficients in parallel. However, it did not balance the cycles consumed by the scan and encode engine, which makes it need (nC+4) cycles to process one 4x4 block where nC equals to the nonzero coefficients in each 4x4 block. Once the nC becomes larger, the throughput of the encoder will be greatly decreased.

In our work, two efforts are made to improve the efficiency:

1) Cycle-Balance: Scan engine and Encode engine are well designed to consume the same clock cycles for each 4x4 block.

2) 4-Coefficient parallel processing: The scan engine generates four coefficients at one cycle. It takes only four cycles to finish the coefficient calculation of one 4x4 block.

3.2 Parallel scan engine

Scan engine is used to generate symbols to be coded. Generally, the coefficients are calculated in reverse zigzag scan order. There are totally 384 coefficients inside one MB, so scan engine may become bottleneck. In order to accelerate scan engine, 4 coefficients are processed in parallel in our design.

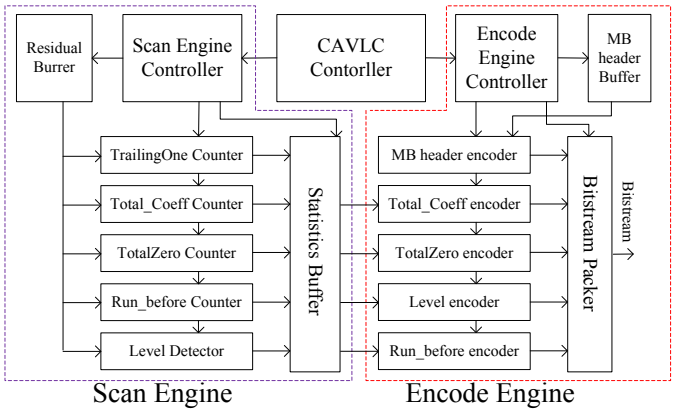

Fig.1 Two-stage architecture of CAVLC.

As shown in equation (1) and (2), s0~s3 are the state bits indicating the coefficient equals to zero. c0~c3 are the state bits indicating the coefficient equals to ±1. For *Total_Coeff*, *TotalZero* and *Run_before*, they can be obtained according to s0~s3. For *TrailingOne*, all of the eight states should be considered together which makes the latency very long.

In order to reduce the *TrailingOne* latency, four coefficients are divided into two parts, the low and high. Each part is calculated independently, and the results are merged finally. The detailed structure is shown in Fig. 2.

$$\begin{cases} if\ coeff\,0 = 0, & s0 = 0 \\ else & s0 = 1 \\ if\ coeff\,1 = 0, & s1 = 0 \\ else & s1 = 1 \\ if\ coeff\,2 = 0, & s2 = 0 \\ else & s2 = 1 \\ if\ coeff\,3 = 0, & s3 = 0 \\ else & s3 = 1 \end{cases} \quad (1)$$

$$\begin{cases} if\ coeff\,0 = \pm 1, & c0 = 1 \\ else & c0 = 0 \\ if\ coeff\,1 = \pm 1, & c1 = 1 \\ else & c1 = 0 \\ if\ coeff\,2 = \pm 1, & c2 = 1 \\ else & c2 = 0 \\ if\ coeff\,3 = \pm 1, & c3 = 1 \\ else & c3 = 0 \end{cases} \quad (2)$$

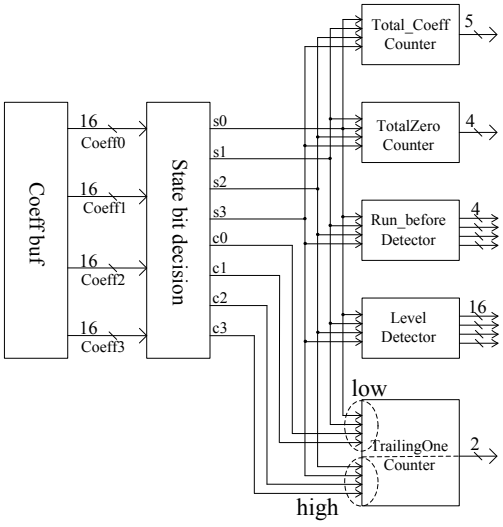

Fig.2 Parallel scan engine structure

For data storage between scan engine and encode engine, it normally uses a ping-pong buffer, such as Chen's [3] work. Since ping-pong buffer needs to store two sets of transaction data, it needs double size of memory, which occupies most of area cost of the CAVLC encoder. For our design, a register array which stores one set of transaction data is adopted to reduce the required buffer size, since the clock cycle of scan engine and encode engine in our design has already been balanced. As a result, it saves 50% buffer size.

3.3 Parallel encode engine

The encode engine encodes coefficients obtained in the scan engine. It consists of *Level* encoder, *Run_before* encoder, *Total_Coeff* and *TrailingOne* encoder, *TotalZero* encoder. *Total_Coeff*, *TrailingOne* and *TotalZero* are coded by lookup tables. The key part of the encode engine is *Level* encoder and *Run_before* encoder.

In H.264/AVC standard, 7 VLC tables are used to define the codeword of *Level* symbols. There are at most 16 *Levels* inside one 4x4 block. The number of execution cycle for *Level* symbols depends on the number of nonzero coefficients in one 4x4 block. Therefore, the number of cycles required for encode stage is various. It may more than or less than that in scan stage. Most of previous works encode the *Level* symbols in serial, because the evaluation of each coefficient *Level* depends on *vlc* [5], which is derived from the previously coded coefficient level, as shown in equation (3). Let *incVLC* ϵ {0, 3, 6, 12, 24, 48, 65535}, *vlc* denotes the index of the VLC table.

$$\begin{aligned} &if\,(vlc = 0) \\ &\quad vlc++; \\ &if\,(abs(level) > incVLC[vlc]) \\ &\quad vlc++; \end{aligned} \tag{3}$$

In order to achieve higher throughput, a parallel *Level* encoding is proposed. The data dependency can be eliminated by using the look-ahead technique so that coefficients can be processed in parallel.

Fig.3 shows the top structure of *Levels* encoder with VLCN look-ahead. In our design, four *Level* symbols are fetched from the Level Register Buffer simultaneously, and four *Level* encoders are used for parallel encoding. Especially, the encode engine will consume the same cycles as the scan engine does. VLCN look-ahead module computes the two *vlc* codes for the corresponding *Level* encoder according to the absolute value of the *Levels* and previous *vlc*. The two *Level* encoders then encode the *Levels* and generate *Level_prefix* and *Level_suffix*. And finally the *Level* codeword packer packs all the four pairs of *Level_prefix* and *Level_suffix* into *FourLevelCode*. The details of *Level* Encoder 0/1/2/3 module are shown in Fig.4.

Similarly, *Run_before* encoder is processed parallel same as *Level* encoder. Two *Run_before* symbols are fetched from Run_before Register array. Two codewords are produced by looking up the *Run_Before&Zerosleft* table. This codewords are packed and stored in a register until all the levels are encoded. The stored codeword is called *TotalZero/Run_before* code and packed by the Bitstream Packer. The detail of the *Run_before* encoder is shown in Fig 5.

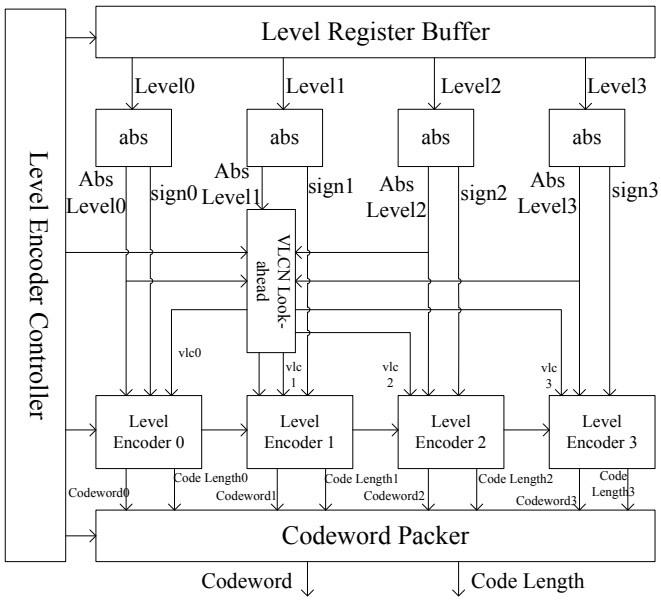

Fig.3 Parallel Level encoder structure.

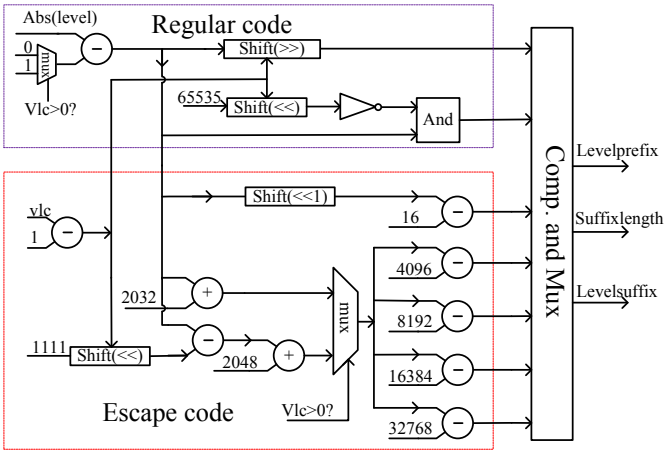

Fig. 4 Detailed structure of Level Encoder 0/1/2/3.

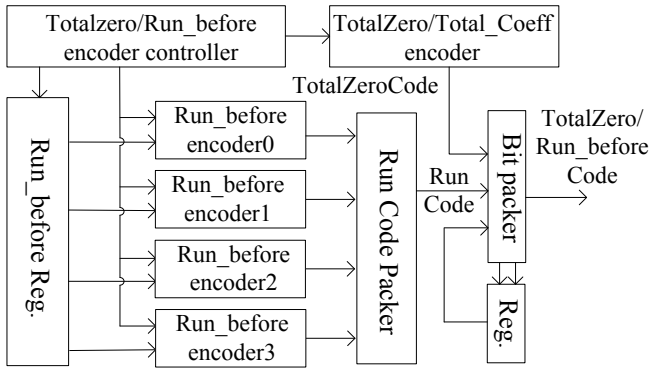

Fig.5 Proposed Parallel Run_before Encoder Architecture

3.4 Bitsteam Packer

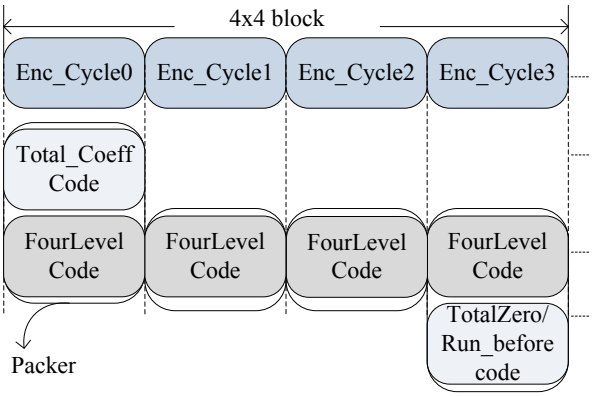

Fig.6 Timing diagram of the Bitstream Packer

The codeword generated by the encoder should be packed into bitstream and this work is done by the Bitstream packer. As mentioned before, all of the codeword should be packed within 4 cycles, so the timing of Bitstream packer should be carefully designed. The detailed timing diagram of Bitstream packer is shown in Fig.6. In Enc_Cycle0, the *Total_coeff* code is packed with the first of *FourLevelCode*. The *Total_Coeff* code can be obtained by looking up the VLC tables according to the value of total *Total_coeff* and *TrailingOne*. The *Four LevelCode* is generated as Fig.3 shows. In Enc_Cycle1~2, the next two *FourLevelCode* are packed. In Enc_Cycle3, the *TotalZero/Run_before* codes are packed with the last *Four LevelCode*.

IV. PEFOFRMACE ANALYSIS AND COMPARISON

The proposed CAVLC hardware design has been implemented by using SMIC 0.13um standard CMOS technology. The total hardware cost is about 32k gates, and the clock frequency is 250 MHz. The hardware cost and performance comparison with the existing designs [3~5] is shown in table I.

For our design, the worst case of throughput is 120 cycles/MB, which is much smaller than previous works [3~5]. It saves more than 50% cycles compare to the state-of-the-art design [5]. In real practice, with the method of coded block pattern (CBP) to determine the blocks that need to be encoded or not, the average cycles will be smaller than the worst case. Fig.7 shows the actual clock cycles consumed under different QP values and video sequences with CBP enabled. According to our experimental result, the average clock cycle under different QPs is about 90.

V. CONCLUSION

This paper proposes a high-throughput CAVLC encoder for high resolution video encoder. The contribution of our work relies on parallel coefficients processing. In order to achieve this parallelism, many methods are proposed, such as clock cycle balance, parallel scan engine and encode engine, statistic buffer size reduction and so on. The implementation results show that our design reduce more than 50% clock cycles, and has ability to support 4096x2160p@60fps video coding in real time.

Table I Comparison with previous works

	Chen[3]	Yi[4]	Hsia[5]	Proposed
Technology (um)	0.18	0.09	0.18	0.13
Frequency	100M	227M	125M	250M
Gate count	23K	66K	15K	32K
speed (cycle/MB)	500	323	256	120
Max. Spec	1920x1080 @30fps	1920x1080 @60fps	1920x1080 @30fps	4096x2160 @60fps
Video format (YUV)	4:2:0	4:4:4	4:2:2	4:2:0

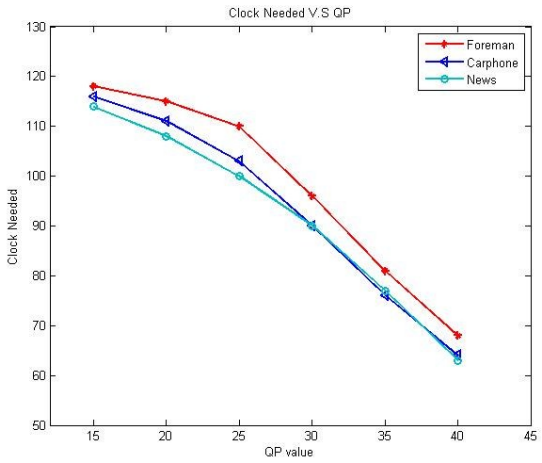

Fig.7 Actual clock cycles used under different QPs and video sequences.

REFERENCES

[1] ITU-T, H.264, Advanced video coding for generic audiovisual services, March 2005.

[2] Joint Video Team (JVT) of ITU-T VCEG and ISO/IEC MPEG, "Joint Model (JM) Reference Software Version 15.1", *http://iphome.hhi.de/suehring*.

[3] T. C. Chen, Y. W. Huang, C. Y. Tsai, B. Y. Hsieh, and L. G. Chen, "Architecture design of context-based adaptive variable-length coding for H.264/AVC," *IEEE Trans. on Circuits Syst. II, Exp. Briefs*, vol. 53, no. 9, pp. 832-836, Sep. 2006.

[4] Y. Yi and B. C. Song, "High-speed CAVLC encoder for 1080p 60-Hz H.264 codec," *IEEE Signal Process. Lett.*, vol. 15, pp. 891–894, 2008.

[5] S.C Hsia and W.S Liao, "Fowward Computations for Context-Adaptive Variable-Length Coding Design," *IEEE Trans. on Circuits Syst. II, Exp. Briefs*, vol.57 no.8, pp. 637-641, Aug 2010.

[6] G. Bjontegaardand K. Lillevold, "Context-adaptive VLC (CAVLC) coding of coefficients," JVT Document JVT-C028. Fairfax, VA,2002.